Department of Electronics and Communications Engineering

Advanced Embedded Systems

ELC4030

CAIRO UNIVERSITY **FACULTY OF ENGINEERING**

EMBEDDED ASSIGNMENT 1

BN	ID	SEC	
4	***9210023***	1	**أحمد اسامه سعد ياسين**

Under supervision of: Dr. Hany El-Sayed

- Rate-Monotonic Scheduling

Part 1

Task 1 (100ms) has the highest priority

Task 2 (200ms) has the medium priority

Task 3 (300ms) has the lowest priority

Setting Tasks outer loop = 10000 and inner loop = 1000, So, The Multiplication of them equals 10×10^6 as required.

First, CURRENT_USING_SCHD has been set into RATE_MONOTONIC.

```
#define RATE_MONOTONIC          1
#define PRIORITY_INVERSION      2

#define CURRENT_USING_SCHD      RATE_MONOTONIC
```

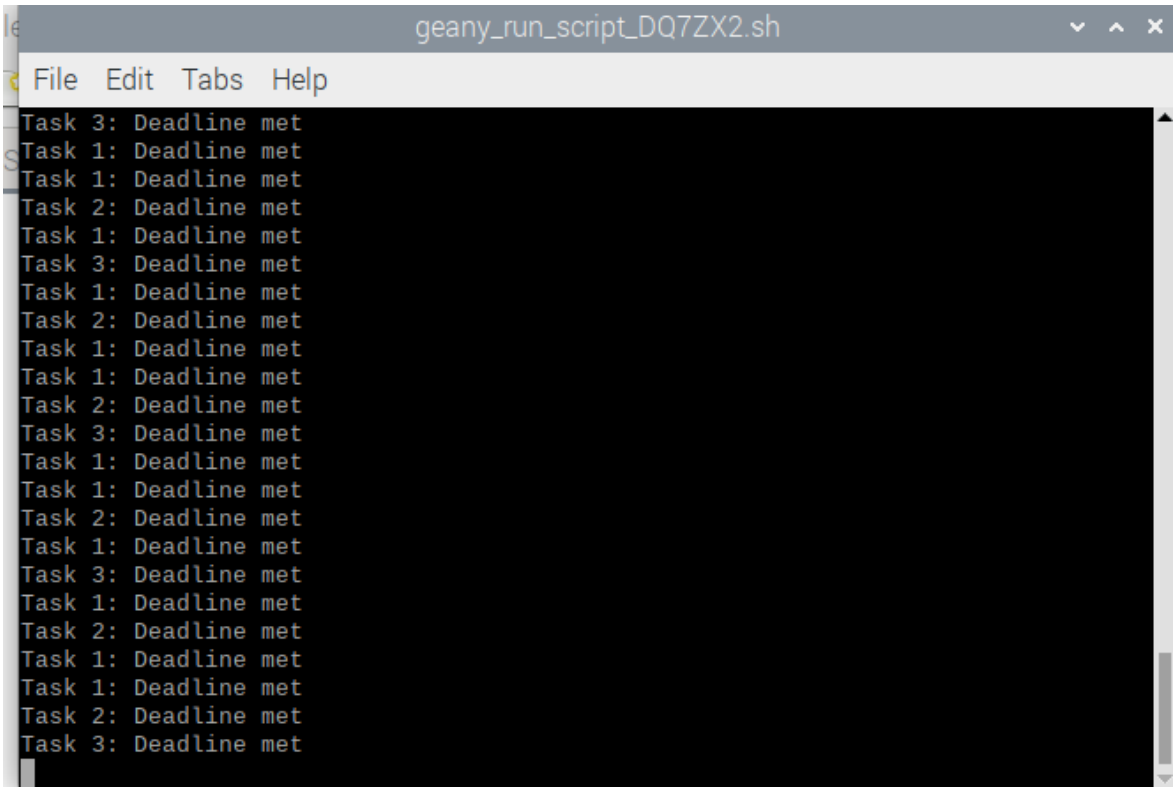
TASK_N1 defines as 10000, and CURRENT_LOOP_CONFIG defines as INNER_LOOP_COUNT_FIXED defines as either INNER_LOOP_COUNT_FIXED as 1000 and INNER_LOOP_COUNT_VARIABLE as 2400

```
#define TASK_N1 10000
// Nested loop iterations (configurable based on workload)
#define INNER_LOOP_COUNT_FIXED 1000
#define INNER_LOOP_COUNT_VARIABLE 2400

// Current workload configuration
#define CURRENT_LOOP_CONFIG INNER_LOOP_COUNT_FIXED
```

TASK_N1 equals to outer loop in the for loop, and CURRENT_LOOP_CONFIG equals into inner for loop.

```
    for (int i=0; i< TASK_N1  i++) {
        for (int j=0; j<CURRENT_LOOP_CONFIG  j++) a=j/2;
        }
```

Results



Discussion

- All tasks met the deadlines as required successfully.

Part 2

First, CURRENT_USING_SCHD has been set into RATE_MONOTONIC.

```
#define RATE_MONOTONIC          1
#define PRIORITY_INVERSION      2

#define CURRENT_USING_SCHD      RATE_MONOTONIC
```

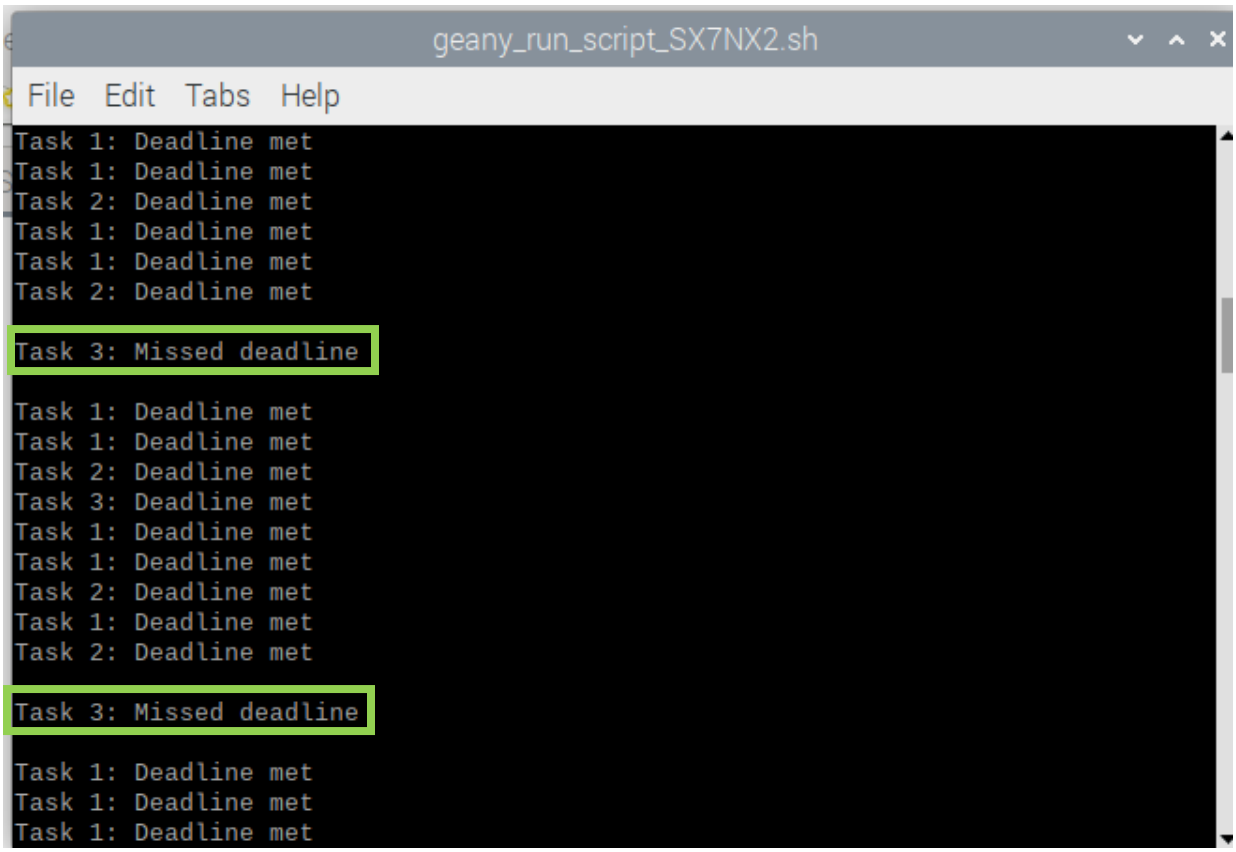
TASK_N1 defines as 10000, and CURRENT_LOOP_CONFIG defines as INNER_LOOP_COUNT_VARIABLE defines as either INNER_LOOP_COUNT_FIXED as 2400

```
#define TASK_N1 10000

// Nested loop iterations (configurable based on workload)
#define INNER_LOOP_COUNT_FIXED 1000
#define INNER_LOOP_COUNT_VARIABLE 2400

// Current workload configuration
#define CURRENT_LOOP_CONFIG INNER_LOOP_COUNT_VARIABLE
```

TASK_N1 equals to outer loop in the for loop, and CURRENT_LOOP_CONFIG equals into inner for loop.

```
    for (int i=0; i< TASK_N1; i++) {
        for (int j=0; j<CURRENT_LOOP_CONFIG; j++) a=j/2;
        }
```

Results



Results Discussion

- As we increase the execution time (increasing the counts of Nested Loops), Task 3 is the first task misses its deadline first as expected

- # Priority Inversion

Part 1

First, CURRENT_USING_SCHD has been set into PRIORITY_INVERSION.

```
#define RATE_MONOTONIC          1
#define PRIORITY_INVERSION      2

#define CURRENT_USING_SCHD      PRIORITY_INVERSION
```

We start adding a mutex lock and unlock in the highest priority task (task 1)

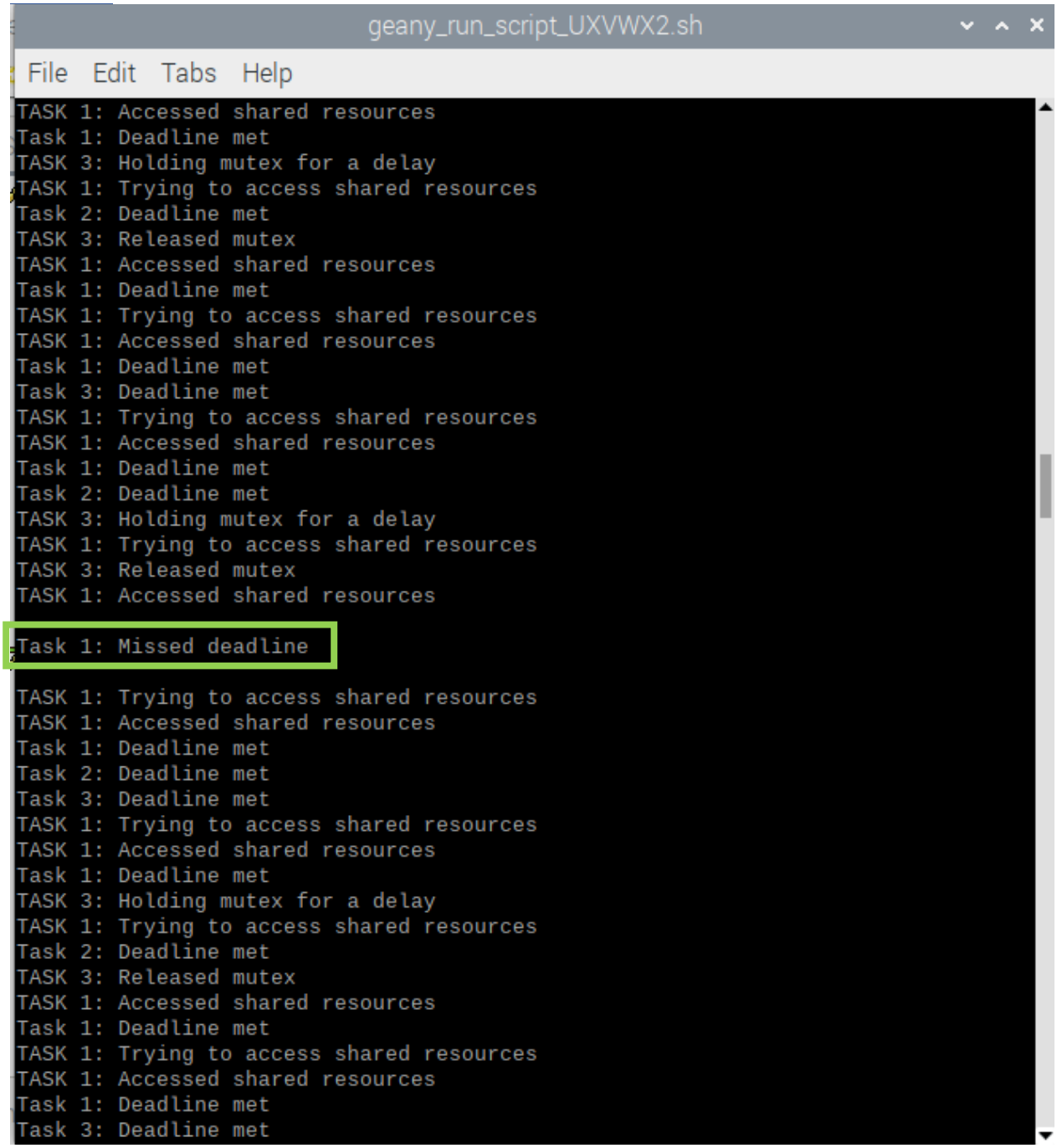
```
#if(CURRENT_USING_SCHD == PRIORITY_INVERSION)
printf("TASK 1: Trying to access shared resources\n");
pthread_mutex_lock(&shared_resource_mutex);
pthread_mutex_unlock(&shared_resource_mutex);
printf("TASK 1: Accessed shared resources\n");
#endif
```

We added the mutex lock and sleep time then mutex unlock in the lowest priority task, and between them we let lowest priority task waits for a long delay 120ms so we use usleep(SLEEP_TIME) → SLEEP_TIME is defined as 120000

```
#if(CURRENT_USING_SCHD == PRIORITY_INVERSION)
pthread_mutex_lock(&shared_resource_mutex);
printf("TASK 3: Holding mutex for a delay\n");
usleep(SLEEP_TIME);
printf("TASK 3: Released mutex\n");
pthread_mutex_unlock(&shared_resource_mutex);
#endif
```

```
#define SLEEP_TIME      120000
```

Results



Single Core Sleep time 120ms – 10 million alliteration.

Results Discussion

- First all tasks are meeting their deadline but most of the time task 3 locks the mutex for a shared resource for 120ms and task 1 wants to access the shared resource at a time task 1 will miss its deadline if task 3 starts before task 1 and holds the mutex for more than task 1 period this will due to a deadline miss.

Part 2

First, I will set the SLEEP_TIME into half (60ms).

```
#define SLEEP_TIME      60000
```

Second, I will set the SLEEP_TIME into double (240ms).

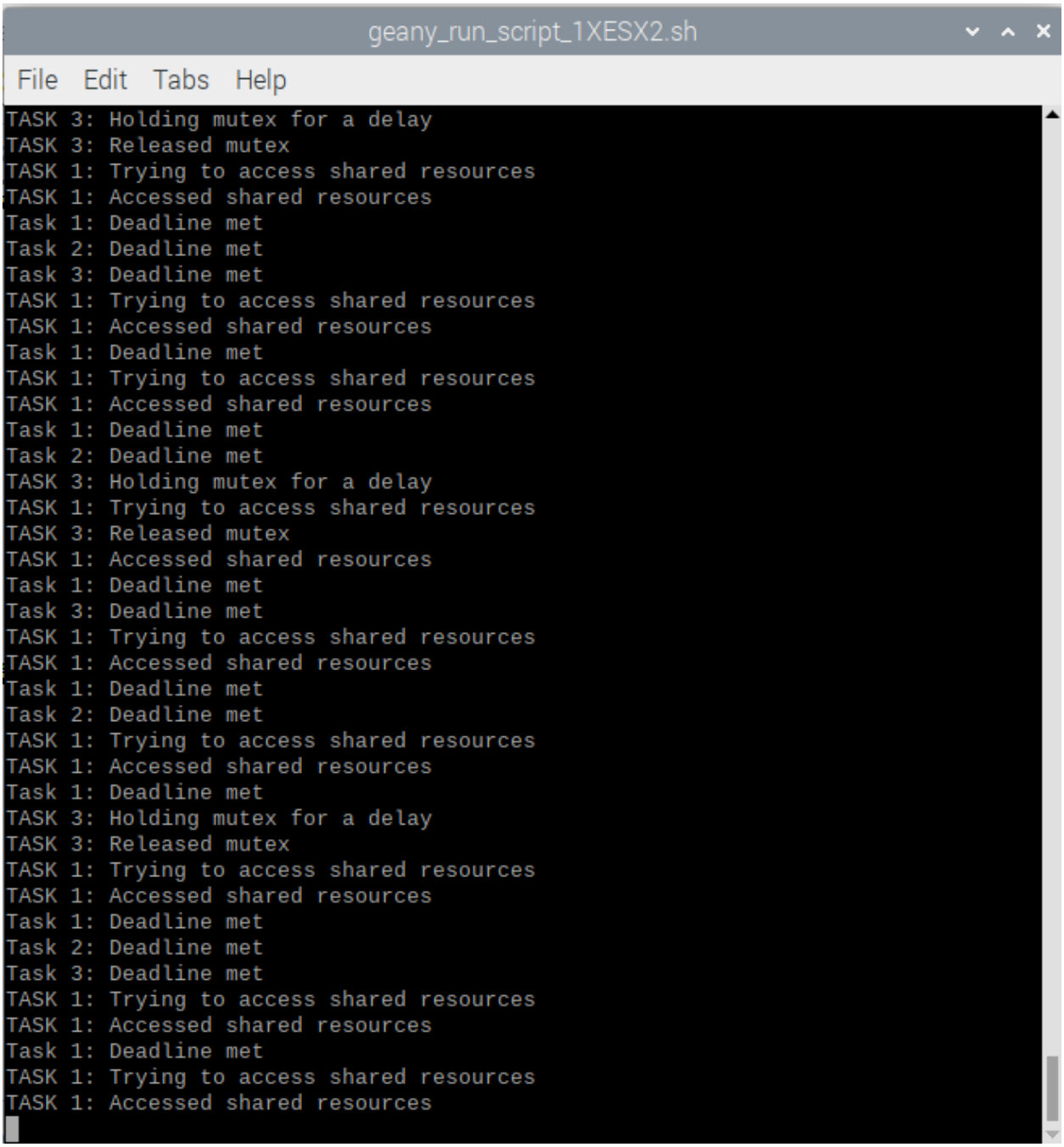
```
#define SLEEP_TIME      240000
```

At the second part of this part, I allow the tasks to run on different cores.

```
#define SINGLE_CORE             1
#define MULTIBLE_CORES          2
#define NUM_OF_CORES            MULTIBLE_CORES
```

```
    #if(NUM_OF_CORES == SINGLE_CORE)
    cpu_set_t cpu_affinity;

    CPU_ZERO(&cpu_affinity);
    CPU_SET(1, &cpu_affinity);

    pthread_attr_setaffinity_np(&attr_task1, sizeof(cpu_set_t), &cpu_affinity);
    pthread_attr_setaffinity_np(&attr_task2, sizeof(cpu_set_t), &cpu_affinity);
    pthread_attr_setaffinity_np(&attr_task3, sizeof(cpu_set_t), &cpu_affinity);

    #elif(NUM_OF_CORES == MULTIBLE_CORES)
    cpu_set_t cpu1, cpu2, cpu3;

    CPU_ZERO(&cpu1);
    CPU_SET(1, &cpu1);
    pthread_attr_setaffinity_np(&attr_task1, sizeof(cpu_set_t), &cpu1);

    CPU_ZERO(&cpu2);
    CPU_SET(2, &cpu2);
    pthread_attr_setaffinity_np(&attr_task2, sizeof(cpu_set_t), &cpu2);

    CPU_ZERO(&cpu3);
    CPU_SET(3, &cpu3);
    pthread_attr_setaffinity_np(&attr_task3, sizeof(cpu_set_t), &cpu3);
    #endif
```

Results 1



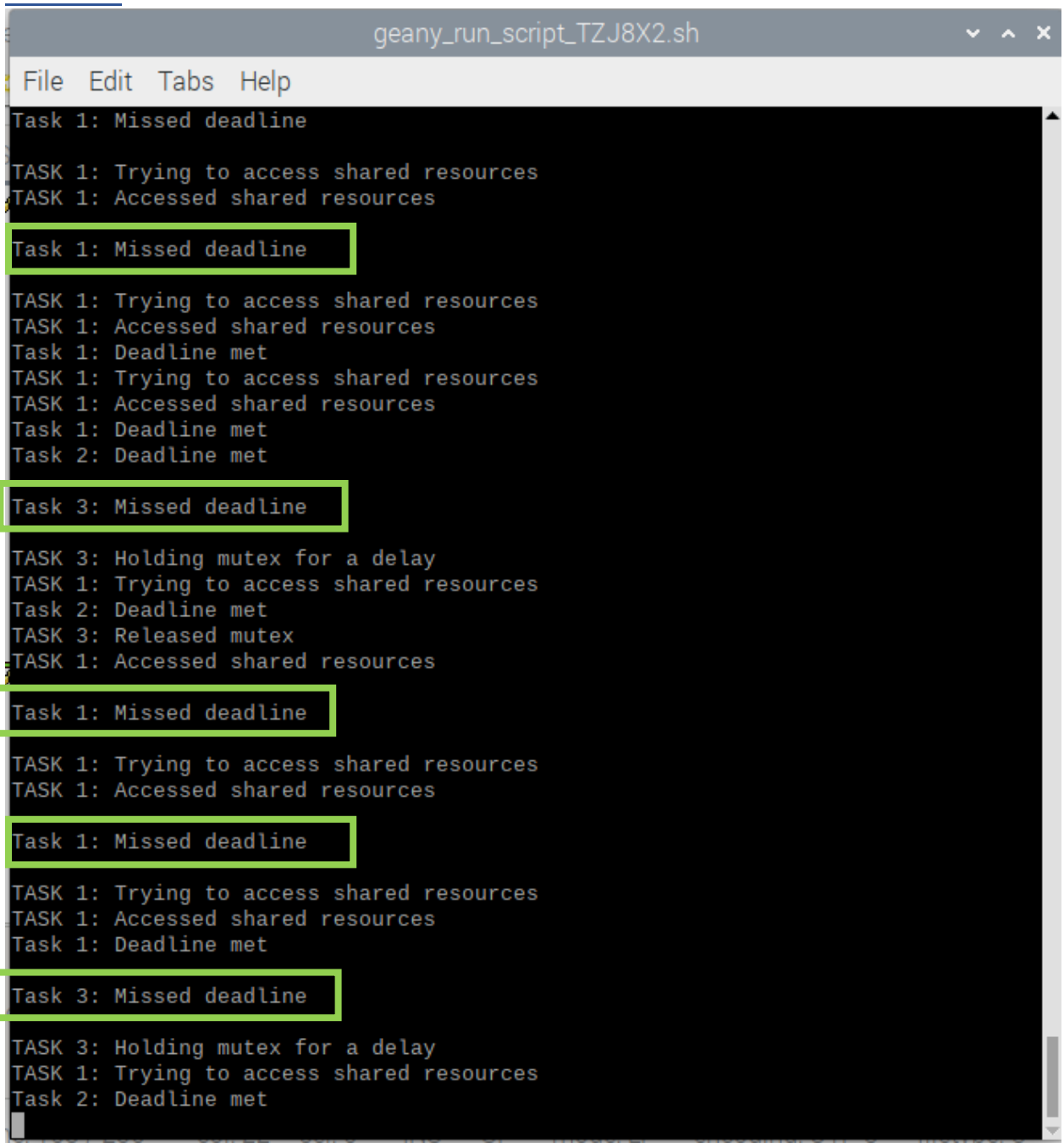
Single Core Sleep time is halved into 60ms – 10 million alliteration.

Discussion

- As the Sleep time decreases the less time lowest priority task taking while having the mutex so it's kind rare that task 1 highest priority will miss its deadline.

  there's no task misses its deadline, tasks met their deadlines.

Result 2



Single Core Sleep time is doubled into 240ms – 10 million alliteration.

Discussion

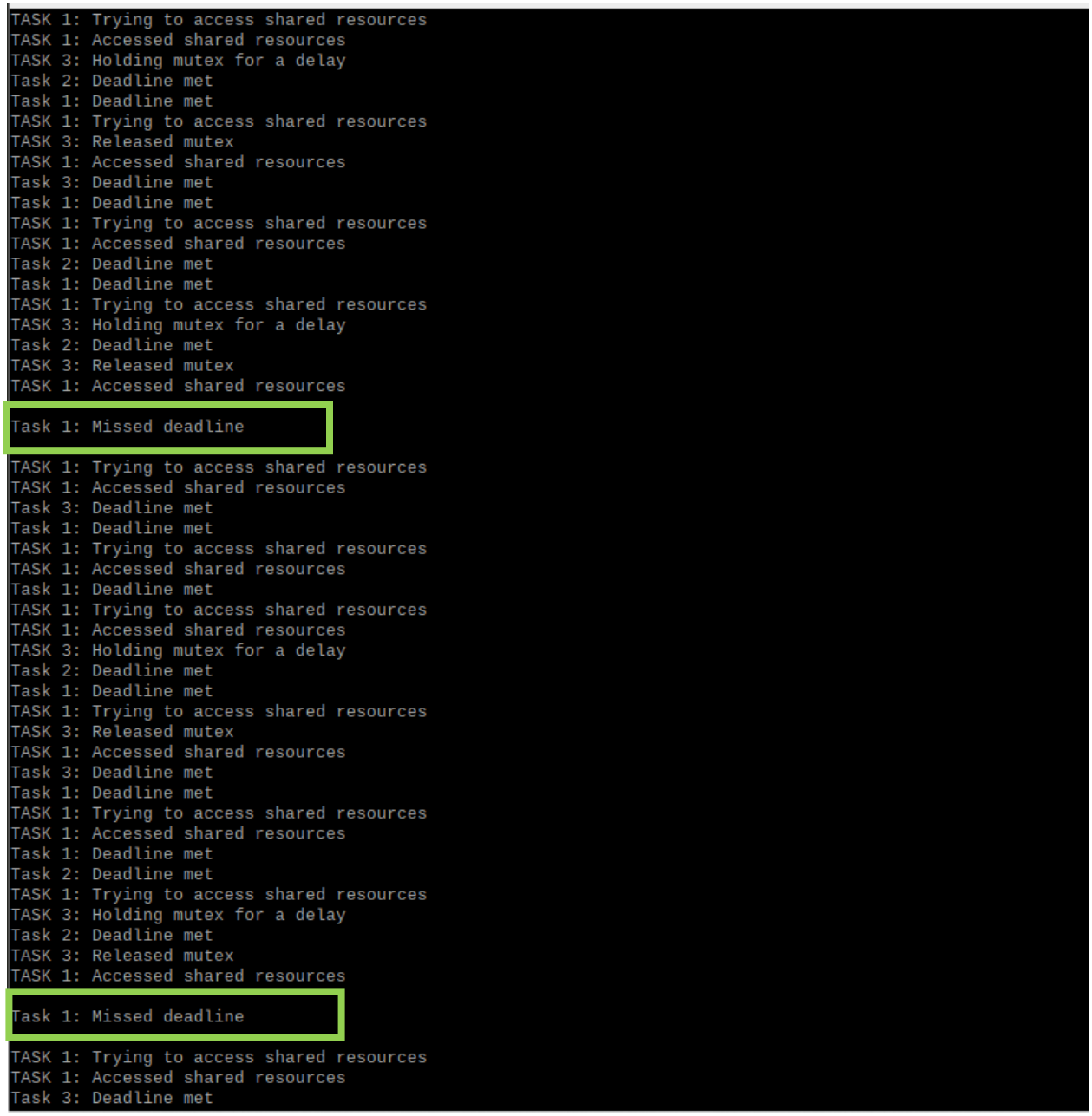
- As the Sleep time doubles the more time lowest priority task taking while having the mutex so it's kind common that task 1 highest priority will miss its deadline. So, the other tasks might miss like task 3.

Results 3

```
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
TASK 3: Holding mutex for a delay
Task 1: Deadline met
Task 2: Deadline met
TASK 3: Released mutex
Task 3: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
Task 2: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
TASK 3: Holding mutex for a delay
Task 1: Deadline met
TASK 3: Released mutex
Task 3: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
Task 2: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
TASK 3: Holding mutex for a delay
Task 1: Deadline met
Task 2: Deadline met
TASK 3: Released mutex
Task 3: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
Task 2: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
TASK 3: Holding mutex for a delay
Task 1: Deadline met
TASK 3: Released mutex
Task 3: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
Task 2: Deadline met
TASK 1: Trying to access shared resources
TASK 1: Accessed shared resources
Task 1: Deadline met
```

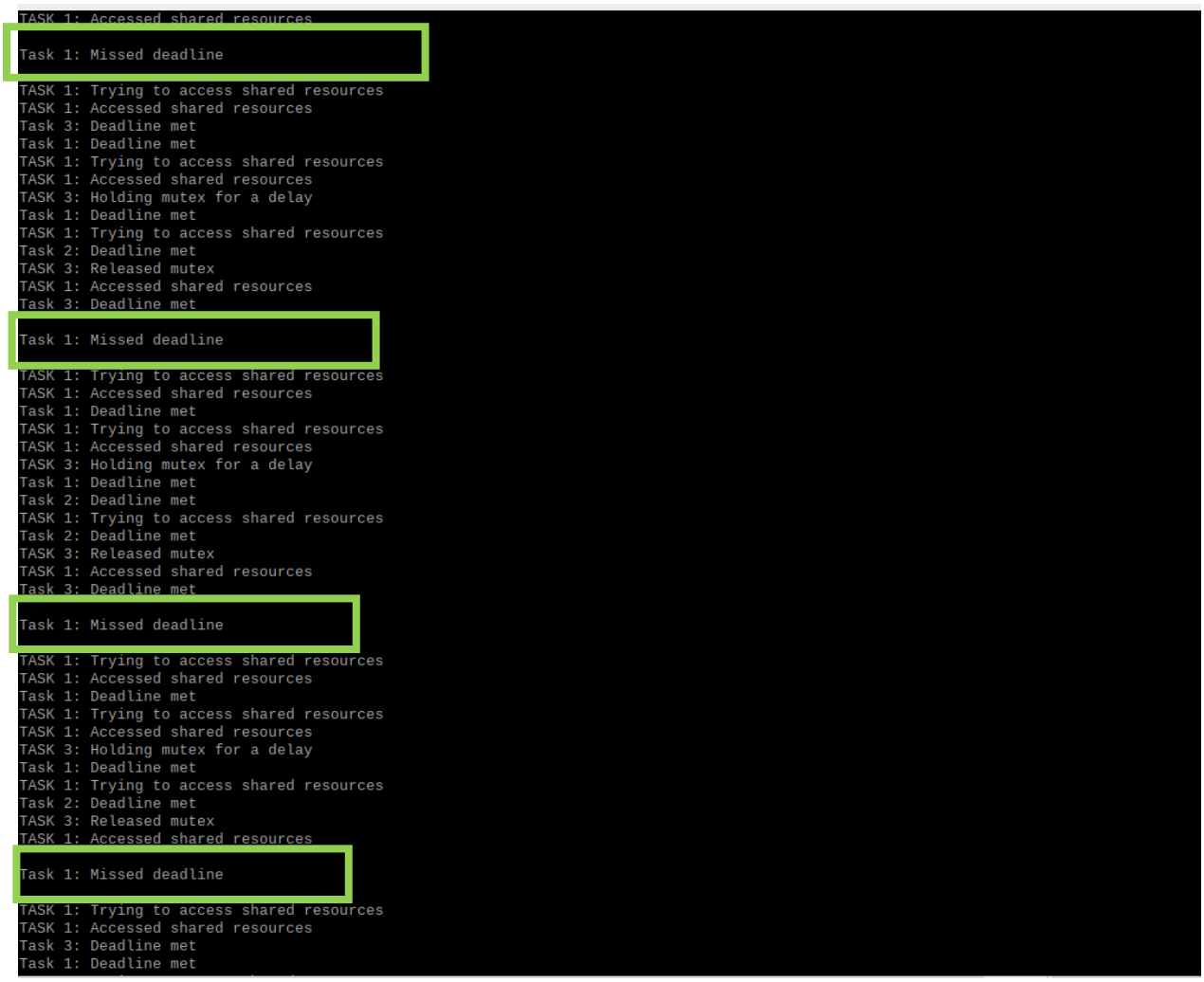
Multi cores – 60ms Sleep – 10 million alliteration.

- There is no difference between the signal core and multi-cores so we are not able to decide which one is the better one. But at this case multi cores it's divided into three cores so the utilization decreases of each one.



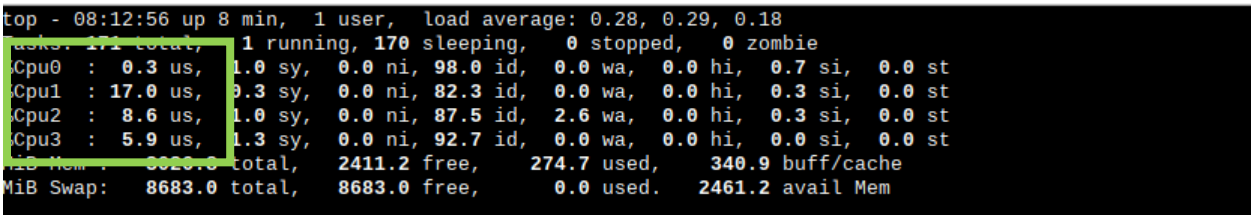
Multi cores – 120ms Sleep – 10 million alliteration.

- there is no big difference between the signal core and multi-cores so we are not able to decide which one is the better one. But at this case multi cores it’s divided into three cores so the utilization decreases of each one.



Multi cores – 240ms Sleep – 10 million alliteration.

- In this case multi cores is much better than single core because its not missing the task 3, because task 3 is running on another cpu so the shared resource is the only problem in this case not the cpu aswell.



CPU Load – Multi cores.

Discussion

- You can notice that the cpu1 has the highest CPU load because it's highest priority and the most period of all tasks then task 2 medium priority in cpu2 and task 3 lowest priority as cp3 lowest CPU load.

2. The Code

```
/*************************************************************************************/
/************************** Author: Ahmed Osama Saad Yassin **************************/
/*************************************************************************************/
/************************************ Includes ***************************************/
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sched.h>
#include <time.h>
#include <math.h>

/************************************ Macros *****************************************/
#define RATE_MONOTONIC                  1
#define PRIORITY_INVERSION              2
#define CURRENT_USING_SCHD              RATE_MONOTONIC
#define SINGLE_CORE           1
#define MULTIBLE_CORES        2
#define NUM_OF_CORES          SINGLE_CORE
#define TASK_N1           10000
// Nested loop iterations (configurable based on workload)
#define INNER_LOOP_COUNT_FIXED 1000
#define INNER_LOOP_COUNT_VARIABLE 2400
// Current workload configuration
#define CURRENT_LOOP_CONFIG INNER_LOOP_COUNT_FIXED
// Constants for time calculations
```

```
#define NANOSECONDS_IN_SECOND 1000000000.0
#define MICROSECONDS_IN_SECOND 1000000.0
#define MILLISECONDS_IN_SECOND 1000.0
// Task periodic intervals in microseconds
#define TASK1_INTERVAL_US 100000  // 100ms
#define TASK2_INTERVAL_US 200000  // 200ms
#define TASK3_INTERVAL_US 300000  // 300ms
// Task priorities
#define TASK1_PRIORITY 3
#define TASK2_PRIORITY 2
#define TASK3_PRIORITY 1
#define MISSED_TIME             1
#define SLEEP_TIME              120000

pthread_mutex_t shared_resource_mutex;
/******************************** Function Prototypes **********************************/
// Utility functions for timespec manipulation
void timespec_add_us(struct timespec *t, long us);
int timespec_cmp(const struct timespec *a, const struct timespec *b);
double subtract_timespecs(const struct timespec *a, const struct timespec *b);

// Task functions
void *task1(void *);
void *task2(void *);
void *task3(void *);
/******************************** Utility Functions **********************************/
// Adds microseconds to a given timespec structure
```

```
void timespec_add_us(struct timespec *t, long us) {
    t->tv_nsec += us * 1000;
    if (t->tv_nsec >= NANOSECONDS_IN_SECOND) {
        t->tv_nsec -= NANOSECONDS_IN_SECOND;
        t->tv_sec += 1;
    }
}
// Compare two timespec structures (-1: earlier, 0: equal, 1: later)
int timespec_cmp(const struct timespec *a, const struct timespec *b){
    if (a->tv_sec > b->tv_sec)
        return 1;
    else if (a->tv_sec < b->tv_sec)
        return -1;
    else if (a->tv_nsec > b->tv_nsec)
        return 1;
    else if (a->tv_nsec < b->tv_nsec)
        return -1;
    else
        return 0;
}
// Calculate the difference in seconds between two timespec structures
double subtract_timespecs(const struct timespec *a, const struct timespec *b){
    double t1 = a->tv_sec + (a->tv_nsec/NANOSECONDS_IN_SECOND);
    double t2 = b->tv_sec + (b->tv_nsec/NANOSECONDS_IN_SECOND);
    if(t1>t2)
            return t1-t2;
    else
            return t2-t1;
}
```

```
/********************************** Task Functions ***********************************/

// Task 1: Periodic execution with resource locking
void *task1(void *args) {
   struct timespec time_current_task, time_next_task = {0};
   int __attribute__((unused)) a =0;
   clock_gettime(CLOCK_REALTIME, &time_next_task);
   while(1){
          #if(CURRENT_USING_SCHD == PRIORITY_INVERSION)
   printf("TASK 1: Trying to access shared resources\n");
          pthread_mutex_lock(&shared_resource_mutex);
          pthread_mutex_unlock(&shared_resource_mutex);
   printf("TASK 1: Accessed shared resources\n");
          #endif
   for (int i=0; i< TASK_N1; i++) {
          for (int j=0; j<CURRENT_LOOP_CONFIG; j++) a=j/2;
   }
      timespec_add_us(&time_next_task, (long)TASK1_INTERVAL_US);
   clock_gettime(CLOCK_REALTIME, &time_current_task);
   if (timespec_cmp(&time_current_task, &time_next_task) == MISSED_TIME)
          printf("\nTask 1: Missed deadline\n\n");
   else
          printf("Task 1: Deadline met\n");
   clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,&time_next_task, NULL);
          }
          return NULL;
}
```

```
// Task 2: Periodic execution without resource locking
void *task2(void *args) {
   struct timespec time_current_task, time_next_task = {0};
   int __attribute__((unused)) a =0;
   clock_gettime(CLOCK_REALTIME, &time_next_task);
   while(1){
   for (int i=0; i< TASK_N1; i++) {
           for (int j=0; j<CURRENT_LOOP_CONFIG; j++) a=j/2;
           }
   timespec_add_us(&time_next_task, (long)TASK2_INTERVAL_US);
   clock_gettime(CLOCK_REALTIME, &time_current_task);
   if (timespec_cmp(&time_current_task, &time_next_task) == MISSED_TIME)
         printf("\nTask 2: Missed deadline\n\n");
   else
         printf("Task 2: Deadline met\n");
   clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,&time_next_task, NULL);
          }
   return NULL;
}

// Task 3: Periodic execution with resource locking and delay simulation
void *task3(void *args) {
   struct timespec time_current_task, time_next_task = {0};
   int __attribute__((unused)) a;
   clock_gettime(CLOCK_REALTIME, &time_next_task);
   while(1){
          #if(CURRENT_USING_SCHD == PRIORITY_INVERSION)
          pthread_mutex_lock(&shared_resource_mutex);
          printf("TASK 3: Holding mutex for a delay\n");
```

```
        usleep(SLEEP_TIME);
    printf("TASK 3: Released mutex\n");
        pthread_mutex_unlock(&shared_resource_mutex);
        #endif
    for (int i=0; i< TASK_N1; i++) {
        for (int j=0; j<CURRENT_LOOP_CONFIG; j++) a=j/2;
          }
    timespec_add_us(&time_next_task, (long)TASK3_INTERVAL_US);
    clock_gettime(CLOCK_REALTIME, &time_current_task);
    if (timespec_cmp(&time_current_task, &time_next_task) == MISSED_TIME)
        printf("\nTask 3: Missed deadline\n\n");
    else
        printf("Task 3: Deadline met\n");
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,&time_next_task, NULL);
        }
    return NULL;
}
int main(){
    pthread_t thread1, thread2, thread3;
    pthread_attr_t attr_task1, attr_task2, attr_task3;
    pthread_mutex_init(&shared_resource_mutex, NULL);
    pthread_attr_init(&attr_task1);
    pthread_attr_init(&attr_task2);
    pthread_attr_init(&attr_task3);
    // Set scheduling policies and priorities
    pthread_attr_setinheritsched(&attr_task1, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setinheritsched(&attr_task2, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setinheritsched(&attr_task3, PTHREAD_EXPLICIT_SCHED);
```

```
pthread_attr_setschedpolicy(&attr_task1, SCHED_FIFO);
pthread_attr_setschedpolicy(&attr_task2, SCHED_FIFO);
pthread_attr_setschedpolicy(&attr_task3, SCHED_FIFO);
#if(NUM_OF_CORES == SINGLE_CORE)
cpu_set_t cpu_affinity;
CPU_ZERO(&cpu_affinity);
CPU_SET(1, &cpu_affinity);
pthread_attr_setaffinity_np(&attr_task1, sizeof(cpu_set_t), &cpu_affinity);
pthread_attr_setaffinity_np(&attr_task2, sizeof(cpu_set_t), &cpu_affinity);
pthread_attr_setaffinity_np(&attr_task3, sizeof(cpu_set_t), &cpu_affinity);
#elif(NUM_OF_CORES == MULTIBLE_CORES)
cpu_set_t cpu1, cpu2, cpu3;
CPU_ZERO(&cpu1);
CPU_SET(1, &cpu1);
pthread_attr_setaffinity_np(&attr_task1, sizeof(cpu_set_t), &cpu1);
CPU_ZERO(&cpu2);
CPU_SET(2, &cpu2);
pthread_attr_setaffinity_np(&attr_task2, sizeof(cpu_set_t), &cpu2);
CPU_ZERO(&cpu3);
CPU_SET(3, &cpu3);
pthread_attr_setaffinity_np(&attr_task3, sizeof(cpu_set_t), &cpu3);
#endif
// Set task priorities
struct sched_param priority_task1 = {.sched_priority = TASK1_PRIORITY};
struct sched_param priority_task2 = {.sched_priority = TASK2_PRIORITY};
struct sched_param priority_task3 = {.sched_priority = TASK3_PRIORITY};
pthread_attr_setschedparam(&attr_task1, &priority_task1);
pthread_attr_setschedparam(&attr_task2, &priority_task2);
pthread_attr_setschedparam(&attr_task3, &priority_task3);
```

```
    // Create tasks
    pthread_create(&thread1, &attr_task1, &task1, NULL);
    pthread_create(&thread2, &attr_task2, &task2, NULL);
    pthread_create(&thread3, &attr_task3, &task3, NULL);
    pthread_attr_destroy(&attr_task1);
    pthread_attr_destroy(&attr_task2);
    pthread_attr_destroy(&attr_task3);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_mutex_destroy(&shared_resource_mutex);
    return 0;
}
```

- README FILE

GitHub Repo → README.md

NOTE: click on GitHub Repo to get into the github repo link. Or click on README.md to get into the readme file directly.